Using the HelpCenter.io API (API Specification)

Integrations

Most days your help center runs on the editor. You write an article, you hit publish, the new piece is live on your subdomain. That covers the daily flow. The cases I want to talk about in this article are the ones that don't fit the editor: pulling a list of every published article into a backup script, mirroring release notes from your engineering tracker the moment they ship, importing a few hundred FAQs from a CSV export, or building a custom search experience that talks to your content from your own product.

Those are jobs for the API. HelpCenter.io exposes a small, focused REST API that lets your code do the things your authors would otherwise do by hand. Read articles. Read categories. Create new articles. Update existing ones. That is the whole surface today, and it is deliberately small. Most teams need exactly those four verbs and nothing more.

This article is the reference. It walks through how to authenticate, what the shared conventions look like, and every endpoint in v1 with a realistic request and response example.

Quick start

You need three things: a HelpCenter.io account, an API key, and somewhere to send a request from.

Open your dashboard, go to **Settings**, scroll to the **API Keys** section. Pick a descriptive name (something like 'Production import script' or 'Release notes sync' is better than 'key1'), choose the permission level, and click **Generate Key**. You can create as many keys as you want.

There are two permission levels. **Read Only** keys can list articles and categories. That covers backups, exports, custom search, content audits, anything where you are pulling data out. **Read & Write** keys can do everything Read Only can do, plus create and

update articles and categories. Use Read Only by default. Issue a Read & Write key only for the specific job that needs it, and rotate it when that job is done.

Keys are tied to a single site. If you run more than one help center under your account (an English instance and a French one, or one site per product), each site has its own keys.

Once you have a token, send it as the apikey header on every request:

```
curl https://api.helpcenter.io/v1/articles \
  -H "apikey: YOUR_API_TOKEN_HERE"
```

Two notes before you build anything serious against this. First, your token is a secret. Treat it like a password. Don't commit it to git, don't paste it in a screenshot, don't email it. If a key leaks, open Settings and delete it. The dashboard lets you revoke any key with one click and existing requests using that token stop working immediately. Second, when you build, build against a staging help center first if you have one. The PATCH and POST endpoints modify real content. There is no soft-delete and no undo button.

Conventions

A few things hold across every endpoint, so you don't have to relearn them per call.

Base URL. Every public API request goes to https://api.helpcenter.io/v1. The v1 is the version prefix. When we ship a v2, the v1 endpoints will keep working on their existing paths. We will not silently change response shapes on you inside a version.

Authentication. One header, apikey: YOUR_TOKEN. We don't use Authorization: Bearer ... and we don't use X-API-Key. Just apikey. Requests without the header, or with an unknown token, get a 401.

Content type. Send Content-Type: application/json on every POST and PATCH. We don't accept form-encoded bodies on the JSON API.

Response envelope. Successful responses are JSON with a status field set to "success". List endpoints add a meta block at the bottom with pagination info. Single-resource endpoints return the resource under a key named after the resource (article, category).

```
{
  "status": "success",
  "articles": [ /* ... */ ],
  "meta": {
    "page": 1,
    "per_page": 100,
    "total_pages": 3,
    "items_count": 247
  }
}
```

Pagination. List endpoints accept ?page= and ?limit= query params. limit defaults to 100 and caps at 100. page defaults to 1. The meta block tells you page, per_page, total_pages, and the absolute items_count, so you can build a loop that stops when page >= total_pages.

Rate limit. The public API allows up to 60 requests per minute per token. That is enough for almost every batch job. If you exceed it, you get a 429 Too Many Requests response and should wait a minute before retrying. If you legitimately need a higher ceiling (a one-time bulk import of thousands of articles, for example), reply to support with what you are building and we will lift the limit for your token.

Multi-language fields. HelpCenter.io is multilingual. Fields like title, slug, content on articles and name, description on categories are stored per language. In API responses they come back as objects keyed by language code (en, de, es, fr, etc.). When you write to those fields, you send the same shape: a JSON object whose keys are the language codes you want to set.

```
{
  "title": {
    "en": "Getting started",
    "de": "Erste Schritte"
  }
}
```

You don't have to send every language. If a key isn't present in the request, the existing translation is preserved. The languages you can use are the ones enabled on your site (your site's default language is always allowed, and any extra languages you have turned on in Settings, Languages).

Error envelope. Errors come back with a non-2xx status code and a JSON body. The exact shape depends on the error class (see the Errors and troubleshooting section at the end of this article), but you can rely on every error being valid JSON with a meaningful HTTP status.

Articles

Articles are the units of content in your help center. Each article belongs to one or more categories, has translations per language, a publish state, and a visibility level. The API gives you four ways to work with them: list, create, update, and updating individual fields again with PATCH.

List articles

Returns a paginated list of articles on your site. You can filter by category, search by keyword, sort by views, and switch the response language.

GET/v1/articles

List the articles on your site. Supports pagination, full-text search, category filter, sort, and per-language responses.

Create an article

Creates a new article on your site. Needs a Read & Write token. The new article is owned by the user the token was generated for, and lives under your site.

A few specifics worth knowing before you call this. title is the only required field, and it has to be a translation object (at minimum your site's default language). If you don't send a slug, one is generated from the title for each language you provided. Setting published: true publishes immediately. Setting published: false (or leaving it off) creates the article as a draft.

If you set visibility: "link_only" the API generates a private share token for you and includes it in the response, so you can build the shareable URL yourself.

POST/v1/articles

Create a new article. Needs Read & Write scope on your token.

A short note on categories vs category_id. The original model has a single category_id. Articles can also be assigned to multiple categories at once, via the categories array (a list of category IDs). If you send categories, the article gets that set of categories and category_id becomes a placeholder value (-1). For most setups, picking one category_id is what you want. Use categories only if you genuinely need an article to appear under several headings.

Update an article

Updates fields on an existing article. Needs a Read & Write token. Send only the fields you want to change. Anything you don't send is left alone, including translations you don't include in the request.

PATCH/v1/articles/{articleId}

Update fields on an existing article. Partial updates are supported. Needs Read & Write scope.

Two quirks worth pre-empting. First, if you send a content field and the article already had content_json cached (from the editor), the cache is dropped so the next edit in the dashboard picks up your new content cleanly. You don't need to do anything special. Second, published: false flips the article back to draft, so it goes off the public site. If you only want to make a quick correction, leave published out of the request.

Categories

Categories are the buckets your articles live under. The API gives you two operations: list them, and update an existing one. There is no public endpoint for creating categories today. We held that one back because creating categories programmatically tends to produce taxonomy sprawl, which is the single most common cause of help centers becoming hard to navigate. Most teams are better off creating categories deliberately in the dashboard and then bulk-importing articles into them via the API.

If you have a genuine use case for creating categories from code (a single-purpose migration script, for example), let us know and we will add the endpoint.

List categories

GET/v1/categories

List the categories on your site. Includes both root-level categories and nested children (use the 'parent' field to walk the tree).

Update a category

PATCH/v1/categories/{categoryId}

Update a category's name, description, parent, or icon. Partial updates are supported. Needs Read & Write scope.

Common workflows

The endpoints above are the building blocks. Here are three things teams actually do with them.

Mirror your release notes into a "What's new" category

The pattern: your engineering team writes release notes in your CMS, your task tracker, or a Markdown folder in your repo. Each time you ship, a small script (a GitHub Action, a webhook from your CMS, or a nightly cron) takes the new entries and posts them as articles in HelpCenter.io under a single category.

Set up a Read & Write token, scope it mentally to 'release notes only', and call POST /v1/articles once per release with category_id set to your "What's new" category and published: true. If you want each entry available in multiple languages, send a title and content map with the languages you have translations for. Anything not translated stays untranslated and the article only appears on the languages where it has content.

For idempotency, store the article ID returned in the 201 response next to the release-notes entry in your source system. The next time the same entry needs updating (a typo fix, an added screenshot), call PATCH /v1/articles/{articleId} instead of creating a duplicate.

Back up everything weekly

The pattern: a scheduled job pulls every article and category from the API and stores the JSON in S3, or in a snapshot table, or in a versioned file in a private GitHub repo. The point is having an export that's independent of the live system.

Use a Read Only token. The flow:

1. Call GET /v1/categories?limit=100&page=1. If meta.total_pages > 1, loop until page >= total_pages.

2. Call GET /v1/articles?limit=100&page=1. Loop the same way.

3. Save the resulting arrays as one JSON file (or one per resource type) with a date stamp.

Most help centers fit comfortably under a few thousand articles, which is a handful of API calls. The whole job typically runs in under a minute and well inside the rate limit.

Import from a CSV or Markdown folder

The pattern: you have content somewhere else (a CSV export from a competitor's tool, a folder of Markdown files in your repo, a Notion export) and you want it inside HelpCenter.io.

The two steps that trip people up: language mapping and category mapping. Decide upfront which language each source row belongs to (most CSV exports don't store this explicitly) and which HelpCenter.io category it should land in. Once you have that, every row becomes one POST /v1/articles call. Send the language code as the key in your title and content objects, set category_id to the matching category, and decide whether to publish on import or stage as drafts.

Two practical tips. First, import as drafts on the first pass so you can review the formatting before publishing. Then iterate by calling PATCH /v1/articles/{articleId} with published: true once each article is ready. Second, do a small batch (5 to 10 articles) end-to-end before you start the full run. The 60 requests per minute limit is generous for a one-shot migration of a few hundred articles, but if you have thousands, sleep briefly between batches or reply on support and we can lift the cap for the duration of your import.

Errors and troubleshooting

The error shapes are simple, but they vary by endpoint. Here is a quick map.

401 Unauthorized comes back when the apikey header is missing, or the token doesn't match a key in our database. Check that the header name is exactly 'apikey' (lowercase, one word) and that you are sending the full token (64 characters, no whitespace). If you recently deleted the key in the dashboard, the token stops working immediately.

403 Forbidden comes back when you try a write operation with a Read Only token. The response message tells you which action was blocked (creating articles, updating articles, updating categories). Generate a Read & Write token in Settings if you need write access.

400 Bad Request is a validation error. The shape varies. GET /v1/articles returns a wrapped envelope with status: 'validation_error' and an 'errors' object. POST and PATCH endpoints return the bare Laravel validation shape: { field: [message] } with no status wrapper. This is a known inconsistency we will harmonise in v2. For now, code defensively and treat any non-2xx as a failure regardless of body shape. An unsupported language code (one that isn't enabled on your site) returns a status: 'error' message.

404 Not Found on PATCH /v1/articles/{id} and PATCH /v1/categories/{id} means no resource with that ID exists on the site the token is associated with. If you are sure the ID is right, double-check that the token is for the correct site (each site has its own keys).

429 Too Many Requests means you have crossed the 60 requests per minute limit on this token. Wait at least one minute and retry. If you regularly need more headroom, reply on support with your use case.

5xx Server Error is on us. If you see one, capture the request ID from the response headers if there is one, plus a copy of the request body that produced it, and email support. We treat these as bugs and they get attention quickly.

What's next

This is the full v1 surface today: list and create articles, update articles, list and update categories. Small on purpose. Most help center automation jobs fit comfortably inside it.

That said, the API will grow as people tell us what they need next. If you hit something the API can't do, that is the most useful thing you can tell us. Reply on the support form with the workflow you are trying to build and the gap you ran into. Our docs and our API

both grow when our customers point at what is missing. What are you building?